4
Preliminares: OWL-S e a Linguagem de Regras do Jena

Este capítulo apresenta duas linguagens necessárias para o entendimento deste trabalho. A Seção 4.1 apresenta uma visão geral de OWL-S [75] abrangendo cada uma de suas ontologias, e a Seção 4.2 apresenta a linguagem de descrição de regras em Jena [62], que é um framework escrito em Java, destinado ao desenvolvimento de aplicações para a Web semântica [13]. A Seção 4.3 apresenta um resumo deste capítulo.

4.1
Ontologia de Serviços de OWL-S

OWL-S, criada com base no conceito de Web semântica, tem como objetivo principal permitir a descoberta, a invocação, a composição e a monitoração automáticas de serviços Web [75].

OWL-S trata composições de serviços como processos. Além disso, em OWL-S existe uma divisão bastante clara entre as propriedades de um processo, o modo como o processo está estruturado e a sua implementação. OWL-S foi selecionada para modelar workflows neste trabalho por oferecer esta visão bastante interessante da modelagem de um processo e independente da tecnologia de implementação, uma vez que separa a estrutura do processo da sua implementação.

4.1.1
Estrutura da Ontologia de Serviços

OWL-S é, sobretudo, uma ontologia de alto nível para a descrição de serviços, mas muitas vezes é chamada de "linguagem", porque ela fornece um vocabulário padrão que pode ser utilizado junto com a linguagem OWL (*Web Ontology Language*) [124] para a descrição dos serviços.

Basicamente, a ontologia de serviços de OWL-S, apresentada na Figura 4.1, representa:

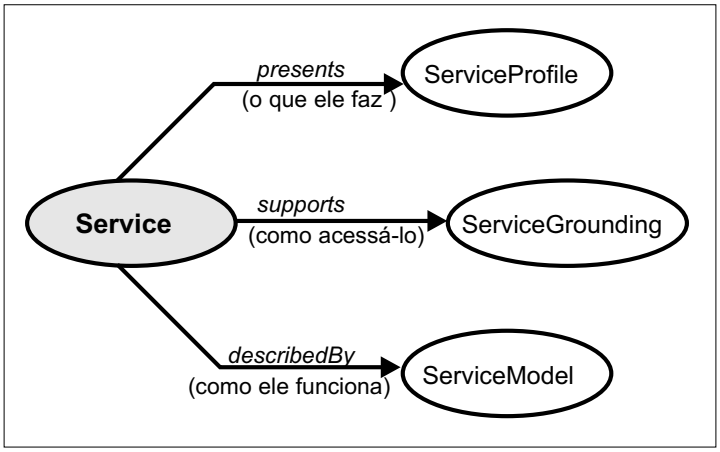

Figura 4.1: Ontologia de serviços de OWL-S, versão 1.1 [75].

– o que o serviço requer do cliente e o que o serviço fornece ao cliente, descrita na classe *ServiceProfile* do serviço. As classes *Service* e *ServiceProfile* são relacionadas em OWL-S através da propriedade *presents* e de sua inversa *presentedBy*;

– como o serviço funciona e quais as conseqüências de sua execução, descritos na classe *ServiceModel*. As classes *Service* e *ServiceModel* são relacionadas em OWL-S através da propriedade *describedBy* e de sua inversa *describes*. Um serviço pode ser descrito por no máximo uma instância de *ServiceModel*;

– como o serviço pode ser acessado, descrito através da classe *ServiceGrounding*. As classes *Service* e *ServiceGrounding* são relacionadas em OWL-S através da propriedade *supports* e de sua inversa *supportedBy*. Uma instância de *ServiceGrounding* associada a uma instância da classe *Service* apresenta os detalhes do protocolo de transporte utilizado pelo serviço especificado. Um *grounding* deve estar associado a exatamente um serviço.

Resumidamente, o *profile* de um serviço Web fornece a informação necessária para um agente descobrir o serviço. Para acessar o serviço, no entanto, o cliente (programa que faz uso do serviço) não utiliza o seu *profile*, mas sim o *grounding* associado ao serviço.

É importante ressaltar que a definição da ontologia de alto nível de serviços, de *profiles*, de *processos* e de *groundings* não limita o desenvolvedor dos serviços a uma única representação. Utilizando os mecanismos de herança de OWL, é possível criar novas representações para serviços, *profiles*,

processos e *groundings*, a partir das ontologias de alto nível definidas em OWL-S.

4.1.2 ServiceProfile

A subontologia de *profiles* de OWL-S está descrita em `http://www.daml.org/services/owl-s/1.1/Profile.owl` e tem como classe principal a classe *ServiceProfile*, que descreve os serviços.

O relacionamento entre a classe *ServiceProfile* e a classe *Service* é bi-direcional, através das propriedades *presents* e *presentedBy*. A propriedade *presents* relaciona uma instância de *Service* com uma instância de *ServiceProfile*, enquanto que sua inversa relaciona uma instância de *ServiceProfile* com uma instância de *Service*.

OWL-S definiu uma subclasse de *ServiceProfile*, a classe *Profile*, como uma possível representação de um perfil de um serviço. Esta representação, segundo OWL-S, contém três propriedades dirigidas ao ser humano, e geralmente não processáveis por máquina: (1) a propriedade *contactInformation*, que descreve o fornecedor do serviço; (2) a propriedade *serviceName*, de cardinalidade igual a 1, usada como um identificador do serviço; e (3) a propriedade *textDescription*, cujo valor é uma descrição textual do que o serviço oferece.

Além dessas propriedades, a classe *Profile* oferece uma descrição das características funcionais de um serviço, sob dois aspectos: a transformação da informação (representada pelas entradas e saídas) e a mudança de estado produzida pela execução do serviço (representada pelas pré-condições e pelos efeitos do serviço).

Em OWL-S existe o conceito de *Parameter*, subclasse da classe *Variable*, definida no contexto da linguagem de regras para a Web semântica denominada SWRL [61], também utilizada em OWL-S. Esta classe *Parameter* é a superclasse tanto da classe que representa os parâmetros de entrada de um processo, *Input*, quanto da que representa os seus parâmetros de saída, *Output*. *Parameter*, *Input* e *Output* são definidas no contexto da ontologia de processos de OWL-S. As pré-condições e os efeitos de um processo, por sua vez, são expressões lógicas e serão explicados em detalhe mais adiante.

Para descrever os parâmetros de entrada e saída, as pré-condições e os efeitos de um serviço, a classe *Profile* conta com as seguintes propriedades, representadas na Figura 4.2, e definidas na ontologia de *profiles*:

– *hasParameter*: propriedade cujo contra-domínio é a classe *Parameter* da ontologia de processos de OWL-S. Esta propriedade não é utilizada diretamente e serve apenas como uma superpropriedade das demais

– *hasInput*: propriedade cujo contra-domínio é a classe *Input*, subclasse de *Parameter*, da ontologia de processos de OWL-S. A classe *Input* representa a informação de que o processo necessita para sua execução;

– *hasOutput*: propriedade cujo contra-domínio é a classe *Output* da ontologia de processos de OWL-S. Tanto *hasOutput* quanto *hasInput* são subpropriedades da propriedade *hasParameter*;

– *hasPrecondition*: propriedade cujo contra-domínio é a classe *Precondition* da ontologia de processos de OWL-S;

– *hasResult*: propriedade cujo contra-domínio é a classe *Result* da ontologia de processos de OWL-S. O resultado de um processo especifica as condições sob as quais as saídas são geradas e, sob estas condições, quais são as alterações causadas pela execução do serviço.

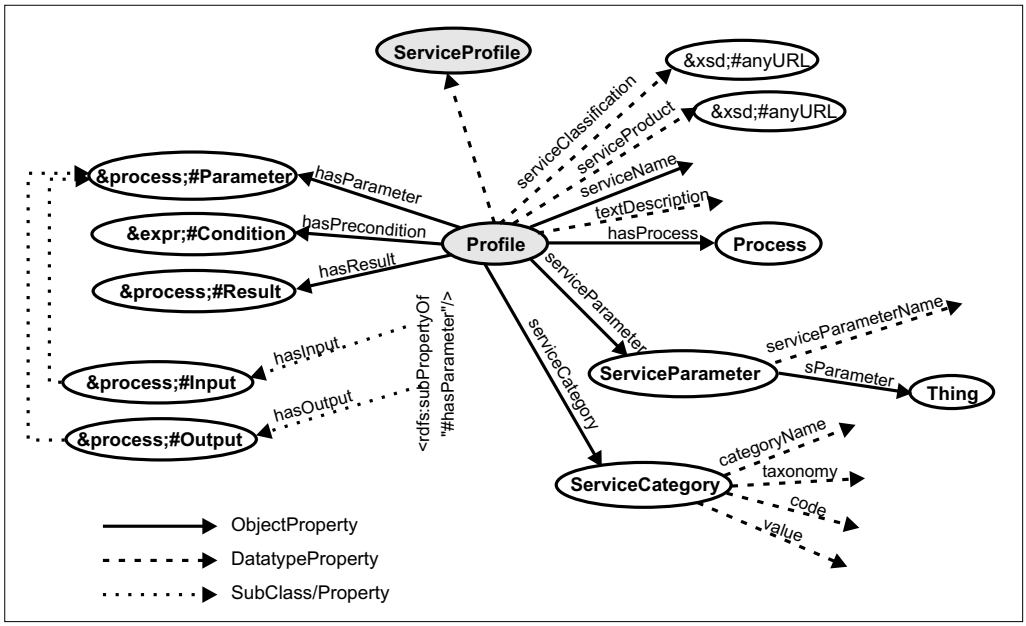

Figura 4.2: Classes e propriedades selecionadas de *ServiceProfile* de OWL-S, versão 1.1 [75].

Idealmente todas as instâncias de *Inputs*, *Outputs*, *Preconditions* e *Results* são criadas na ontologia de processos e referenciadas na classe

Profile. Desta forma, cada uma destas propriedades possuem como valor, não novas instâncias criadas na instância da classe *Profile*, mas sim uma referência para instâncias já criadas na ontologia de processos do serviço correspondente.

Por fim, a classe *Profile* apresenta um conjunto de características específicas de um serviço, por meio da propriedade:

– *serviceParameter*: uma lista estensível de propriedades, sendo que cada valor da propriedade é uma instância da classe *ServiceParameter*. Esta classe *ServiceParameter* possui as propriedades *serviceParameterName*, que indica o nome do parâmetro, e *sParameter*, que aponta para o valor do parâmetro em alguma ontologia OWL. Esta propriedade permite que diversos outros atributos sejam associados a um serviço, de acordo com a funcionalidade que ele deseja assumir; e

– *serviceCategory*: se refere a uma entrada em alguma ontologia ou taxonomia de serviços existente, sendo o seu valor uma instância da classe *ServiceCategory*. Esta classe descreve categorias de serviços e possui um nome (propriedade *categoryName*), aponta para uma taxonomia (propriedade *taxonomy*), possui um ou mais valores na taxonomia especificada (propriedade *value*) e, para cada tipo de serviço, possui um código associado na taxonomia (propriedade *code*).

Note que um *Profile* pode tornar públicas apenas as funcionalidades do serviço que o seu provedor desejar, independentemente de quantas outras funcionalidades o serviço for capaz de atender. No entanto, uma vez não declaradas, estas funcionalidades nunca serão encontradas e, por isso, não poderão ser utilizadas.

4.1.3 ServiceModel

Em OWL-S, um serviço é considerado um processo, quando visto de forma detalhada, e este processo é descrito por meio da classe *Process*, subclasse de *ServiceModel*. Esta é a classe mais importante sob o ponto de vista do trabalho aqui proposto, pois é a partir dela que workflows são especificados.

Processos possuem a eles vinculadas condições que determinam situações sob as quais podem ser executados. Processos também podem

ter um conjunto de entrada de dados e a execução do processo pode levar à transformação destes dados em um conjunto de dados de saída. Uma execução também pode provocar "transformações no mundo", de um estado para outro, descritas através dos *efeitos* do processo. Por exemplo, em um processo de compra de um produto, o efeito pode ser o débito do valor do produto da conta corrente do comprador e o seu crédito na conta corrente do vendedor. A saída do processo, por sua vez, pode ser uma notificação de que o processo de compra foi efetuado com sucesso.

Dessa forma, para se descrever um processo em OWL-S 1.1, é necessário especificar seu conjunto de parâmetros de entrada, seu conjunto de parâmetros de saída, suas pré-condições e seus efeitos. Além disso, é preciso definir o seu fluxo de dados, ou seja, quais entradas de subprocessos são provenientes de saídas de subprocessos anteriormente executados.

Assim como o *ServiceProfile*, o *ServiceModel* é uma representação do serviço. No entanto, estas classes são diferentes, mas devem garantir que, para um mesmo serviço, as IOPEs (*Input*, *Output*, *Precondition* e *Effect*s) definidas em uma representação devem ser refletidas na outra representação.

4.1.3.1. Parâmetros, Resultados e Expressões de um Processo

Em OWL-S, tanto *Inputs* quanto *Outputs* são subclasses de *Parameter*, conforme já mencionado. Cada parâmetro possui um tipo, identificado pela propriedade *parameterType*, que indica a classe à qual seus valores pertencem. O seu escopo é o processo dentro do qual está definido. Além da definição do tipo do parâmetro, se o seu valor é uma constante, então ele pode ser definido em OWL-S através da propriedade *parameterValue*. Esta é a forma mais direta de se associar um valor constante a um parâmetro de um processo, embora também seja possível através de *valueData*, a ser explicada mais adiante.

OWL-S define um outro tipo de parâmetro, chamado de *Local*, também subclasse de *Parameter*, e que, uma vez definido, pode ser usado em qualquer lugar do processo. Variáveis locais apenas podem ser usadas em processos atômicos.

Para descrever o relacionamento entre um processo e cada um de seus parâmetros, tornando possível a especificação da transformação de dados, cada processo OWL-S possui as propriedades definidas na ontologia de processos *hasInput*, *hasOutput* e *hasLocal*, subpropriedades da propriedade *hasParameter*, cujos contra-domínios são, respectivamente, as classes *Input*,

Output e *Local*. Não existe razão para que a propriedade *hasParameter* da ontologia de processos seja diretamente utilizada; ela pode ser comparada a uma classe abstrata em Java e serve apenas como superpropriedade das propriedades já citadas.

Além de parâmetros de entrada e de saída, processos em OWL-S podem estar associados a pré-condições, de modo que apenas são executados quando estas pré-condições se tornam verdadeiras. A propriedade de *Process* *hasPrecondition* indica a sua pré-condição de execução.

Quando um processo é executado, além dos dados de saída, os *Outputs*, ele pode gerar alterações no estado do mundo, chamadas de *Effect*. Conforme já mencionado, efeitos são diferentes de dados de saída porque não são dados gerados pelo processo, mas são alterações produzidas pela execução do processo sobre algum objeto no mundo, como por exemplo um conta corrente quando do recebimento de um dinheiro, uma parede quando do recebimento de uma mão de tinta, dentre outros exemplos.

Para expressar as pré-condições e os efeitos de um processo, OWL-S utiliza expressões em uma linguagem específica. Assim, *Condition* e *Effect* são subclasses de *Expression*, descrita em (`http://www.daml.org/services/owl-s/1.1/generic/Expression.owl`). Esta superclasse, por sua vez, possui duas propriedades de cardinalidade igual a 1: *expressionLanguage*, que identifica em qual linguagem a expressão está escrita, e *expressionBody*, que contém a expressão em si. Uma das linguagens mais citadas para representação das condições em OWL-S é a SWRL.

Em OWL-S, *Outputs* e *Effects* não são diretamente relacionados aos processos aos quais dizem respeito, uma vez que esta ligação pode ser dependente do contexto. Neste sentido, na versão 1.1 da linguagem foi criada a classe *Result*, que mantém uma relação direta com a classe *Process* através da propriedade *hasResult*. Assim, o resultado de um processo em OWL-S pode conter tanto os dados de saída quanto os efeitos gerados pela execução deste processo.

A classe *Result* possui cinco propriedades. A propriedade *hasResultVar* permite a declaração de uma variável, da classe *ResultVar*, subclasse de *Parameter*, que tem seu escopo limitado ao resultado dentro do qual está sendo declarada. Uma vez declarada, esta variável pode ser usada para descrever as saídas e os efeitos daquele processo, sob a condição definida para o resultado no qual está contida. Esta condição é definida pela propriedade *inCondition*, cujo contra-domínio é a classe *Condition*.

A propriedade opcional *withOutput*, por sua vez, com contra-domínio em *OutputBinding*, especifica as saídas do processo quando da ocorrência

daquele resultado. A propriedade *hasEffect*, analogamente, especifica os efeitos da execução deste processo sob este resultado. Finalmente, existe uma propriedade associada a *Result*, chamada *resultForm*, cujo objetivo é fornecer um template XML abstrato para as saídas enviadas de volta ao cliente do serviço.

Vale ressaltar que mais do que um resultado pode ser definido para um mesmo processo OWL-S. No entanto, as condições sob as quais eles são válidos devem ser mutuamente exclusivas, de modo que apenas um resultado ocorra a cada execução do processo.

Uma outra característica de processo em OWL-S é que ele envolve no mínimo dois agentes: o cliente, representado pela classe *TheClient*, aquele que faz uso do serviço, e o servidor, representado pela classe *TheServer*, aquele que implementa e oferece o serviço.

4.1.3.2. Fluxo de Dados de um Processo

Em processos compostos, a entrada de um processo componente (ou subprocesso) pode ser obtida da saída de processos componentes anteriores. Outro tipo de fluxo de dados é a especificação das saídas de um processo composto como derivações das saídas de alguns de seus processos componentes. No entanto, a questão do fluxo de dados não está claramente definida em OWL-S. Como será visto no Capítulo 7, assumimos que se um processo p recebe valores de entrada provenientes de parâmetros de um outro processo p' então p e p' possuem um superprocesso em comum, ou seja, pertencem à mesma hierarquia de processos.

Em OWL-S 1.1 foi adotada a convenção *consumer-pull* para a especificação do fluxo de dados. De acordo com esta convenção, a fonte de um dado apenas é identificada quando o usuário (consumidor) do dado é declarado, e não quando o processo ao qual pertence é declarado.

Assim, foi criado o conceito de *binding*, através da classe *Binding*. Este conceito é então utilizado para: (1) determinar valores para os parâmetros de entrada de um processo, dentro da sua definição; e (2) determinar como valores de parâmetros de saída são especificados dentro do resultado do processo. *InputBinding* e *OutputBinding* são subclasses de *Binding*, onde *InputBinding* tem como valor da propriedade *toParam* uma instância de *Input* e *OutputBinding* uma instância de *Output*.

Associadas à classe *Binding* existem duas propriedades: *toParam*, que identifica o nome do parâmetro, e *valueSpecifier*, que corresponde

à descrição de seu valor. Esta especificação de valor pode ocorrer de quatro formas distintas (subpropriedades de *valueSpecifier*): *valueSource*, *valueType*, *valueData* e *valueFunction*, dentre as quais as mais utilizadas são *valueSource* e *valueType*.

A propriedade *valueSource* possui como contra-domínio a classe *ValueOf*, que especifica as propriedades *theVar*, cujo contra-domínio é a classe *Parameter*, e *fromProcess*, com contra-domínio *Perform*. Esta propriedade permite indicar que o valor deste parâmetro é proveniente do valor de um parâmetro em outro processo, no mesmo processo composto. Desta forma, se um *Binding* com $toParam = p$ tem $valueSource = s$ com propriedades $theVar = v$ e $fromProcess = R$, isto significa que o parâmetro p do processo é igual ao parâmetro v de R.

A propriedade *valueType* tem como valor uma URI (*Uniform Resource Identifier*) que aponta para uma classe OWL, indicando que o valor do parâmetro pertence a esta classe. A classe especificada deve ser subclasse da classe especificada em *parameterType* do referido parâmetro.

A propriedade *valueData* serve para a especificação de constantes em XML. A propriedade *valueFunction* representa funções parametrizadas e foi projetada com o objetivo de permitir futuras extensões da linguagem.

Assim, a propriedade *hasDataFrom* de *Process* é utilizada para relacionar *Inputs* de *Performs* a *InputBindings*, indicando de onde vêm os valores dos parâmetros de entrada do processo. A propriedade *withOutput* de *Result*, por sua vez, que tem como contra-domínio a classe *OutputBinding*, permite que o valor de um parâmetro de saída seja especificado dentro de um resultado, de acordo com a condição a ele associada.

A variável *TheParentPerform* foi introduzida em OWL-S para se referir, em tempo de execução, a uma particular instância em execução de um processo definido via *Perform*, como componente de um processo composto. Assim, torna-se possível referenciar o valor de um determinado parâmetro durante uma execução.

4.1.3.3 Composição dos Processos

A classe *Process* de OWL-S (`http://www.daml.org/services/owl-s/1.1/Process.owl`) corresponde à união das três classes a seguir:

– *AtomicProcess*: representa processos que podem ser diretamente invocados, que não têm subprocessos e que executam em um único passo sob a perspectiva do cliente. Para cada processo atômico deve

existir um *grounding* associado. *TheClient* é o valor de sua propriedade *hasClient* e *TheServer* o valor de sua propriedade *performedBy*;

– *SimpleProcess*: representa processos que não podem ser diretamente invocados, mas que, assim como um processo atômico, são vistos como executados em apenas um único passo. Processos simples são utilizados como elementos de abstração. Eles podem ser usados para fornecer uma visão de um processo atômico ou uma representação simplificada de um processo composto. Se um processo simples está vinculado a um processo atômico, este vínculo se dá por meio da propriedade *realizedBy*. Se um processo simples está vinculado a um composto, então este vínculo ocorre por meio da propriedade *expandsTo*. Processos simples podem ter a eles associados parâmetros de entrada e saída, assim como os demais tipos de processos. Este tipo de processo ainda não recebeu a atenção merecida pela comunidade desenvolvedora e que utiliza a linguagem OWL-S;

– *CompositeProcess*: representa processos que podem ser decompostos em outros processos. Processos compostos têm suas composições descritas através dos construtores de controle (ou *ControlConstruct*), por meio da propriedade *composedOf*, cuja cardinalidade é igual a 1. Cada *ControlConstruct*, por sua vez, possui uma propriedade denominada *components*, que indica como é a composição do processo.

Um processo composto, o mais complexo dos processos, pode ser considerado uma árvore cujos nós não-terminais são rotulados com construtores de controle, cada um dos quais tendo seu filho especificado usando a propriedade *components*. As folhas da árvore representam a invocação de outros processos, indicadas como instâncias da classe *Perform*. A propriedade *process* da classe *Perform* indica o processo (classe *Process*) a ser invocado.

A Figura 4.3 apresenta a ontologia de processos definida em OWL-S 1.1.

OWL-S inclui os seguintes construtores de controle: *Sequence*, *Split*, *Split-Join*, *Any-Order*, *Choice*, *If-Then-Else*, *Iterate*, *Repeat-While* e *Repeat-Until*. A semântica dos construtores de controle em OWL-S é similar à semântica dos respectivos construtores de controle em BPEL, BPML ou qualquer uma das outras linguagens para a modelagem de processos. Note,

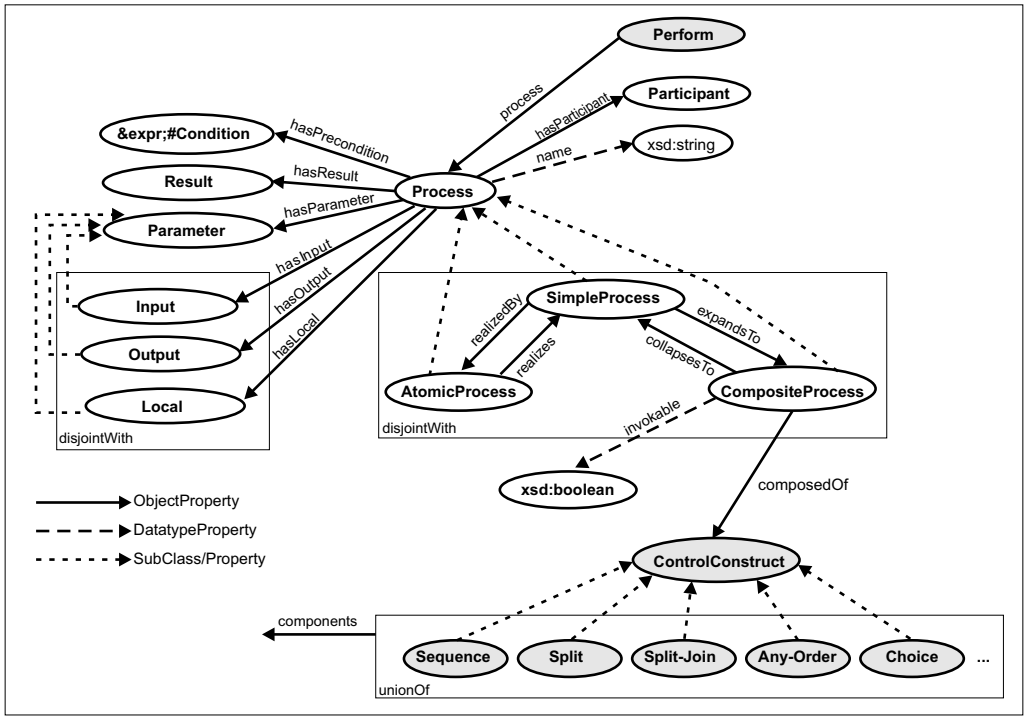

Figura 4.3: Ontologia de processos de OWL-S, versão 1.1 [75].

porém, que não há uma semântica formalmente definida para OWL-S e que esteja oficialmente publicada.

4.1.4 ServiceGrounding

A informação de como acessar um serviço em OWL-S é descrita na classe *ServiceGrounding*, definida em `http://www.daml.org/services/owl-s/1.1/Grounding.owl`. Esta informação diz respeito basicamente ao formato das mensagens que devem ser trocadas e aos protocolos de serialização, transporte e endereçamento.

Um *grounding* em OWL-S pode ser visto como um mapeamento de uma especificação abstrata para uma especificação concreta dos elementos que descrevem o serviço e que são necessários para interagir com o serviço. Assim, dentre os documentos que juntos formam a linguagem OWL-S, o *ServiceGrounding* é o único que lida com um nível concreto de especificação, enquanto o *ServiceProfile* e o *ServiceModel* são representações em um nível abstrato.

No entanto, OWL-S não inclui uma construção abstrata para explicitamente descrever mensagens. O que ocorre, de fato, é que o conteúdo abstrato de uma mensagem é implicitamente especificado pelas propriedades de entrada e de saída dos processos atômicos. As mensagens concretas, ao

contrário, são explicitamente especificadas em um *grounding*. Para isso, é utilizada a linguagem WSDL (*Web Services Description Language*), já citada no Capítulo 3. O objetivo do *grounding* então é determinar como as entradas e saídas de um processo atômico podem ser descritas em termos de mensagens trocadas entre os processos.

O conceito de *grounding* em OWL-S é geralmente consistente com o conceito de *binding* de WSDL. Porém, as duas linguagens são necessárias para a especificação concreta de um processo, porque elas não cobrem o mesmo espaço conceitual.

O *grounding*, por tratar propriamente de questões de implementação, é o conceito menos importante no contexto do trabalho aqui proposto e, por isso, não será descrito em detalhes.

Maiores informações sobre a linguagem e comentários adicionais a respeito de terminologia e expressividade podem ser encontrados na lista de discussão do W3C, acessível através da URL `http://lists.w3.org/Archives/Public/public-sws-ig/`.

4.1.5 Ontologia de Recursos

Além da ontologia de serviços, de *profiles* e de *groundings*, OWL-S ainda propõe uma ontologia de recursos (`http://www.daml.org/services/owl-s/1.1/Resource.owl`), uma vez que processos geralmente necessitam de recursos para a execução. No entanto, a ontologia de recursos de OWL-S não faz parte da submissão ao W3C, datada de novembro de 2004.

Em OWL-S, um recurso tem uma propriedade denominada *allocationType*, que designa qual o tipo de alocação do recurso. Um recurso é dito consumível (*ConsumableAllocation*) se ele não mais existir depois que a atividade que o utilizou ter terminado. Caso contrário, o recurso é dito reusável. Se um recurso é reusável (*ReusableAllocation*) ou persistente, ele pode ser bloqueado e liberado posteriormente. Quando ele é bloqueado para utilização por algum processo ou alguma atividade, ele não pode ser utilizado por nenhum(a) outro(a) até que seja liberado. Vale ressaltar que a ontologia de OWL-S não armazena informação de bloqueio de recursos.

Um recurso OWL-S também possui uma propriedade denominada *capacityType*, que pode ser do tipo *ContinuousCapacity* ou *DiscreteCapacity*. Quantidade de combustível pode ser vista como tendo uma capacidade contínua, enquanto um número de cadeiras pode ser vista como tendo uma capacidade discreta.

A capacidade de um recurso está associada a uma granularidade, representada em OWL-S pela propriedade *capacityGranularity*, que indica a unidade em termos da qual a capacidade é medida. Por exemplo, o recurso "combustível" pode ser medido em litros.

Ainda, um recurso pode ser atômico (*AtomicResource*) ou agregado (*AggregateResource*), sendo um recurso agregado um conjunto de recursos alocados para a execução de uma atividade.

Muitas vezes um recurso atômico pode ser utilizado apenas por um único processo em determinado instante de tempo (recurso está disponível ou não). Esse tipo de recurso é chamado em OWL-S de *UnitCapacityResource*. Por outro lado, *BatchCapacityResource* é um recurso que pode ser compartilhado por diversas atividades, de modo sincronizado.

Recursos agregados podem ser conjuntivos (*ConjunctiveAggregateResource*), de modo que todos os elementos devem ser alocados para a atividade, ou disjuntivos (*DisjunctiveAggregateResource*), de modo que apenas um subconjunto de elementos pode ser alocado para a atividade.

A Figura 4.4 apresenta a ontologia de recursos de OWL-S, gerada pelo plugin ezOWL do Protégé 3.0.

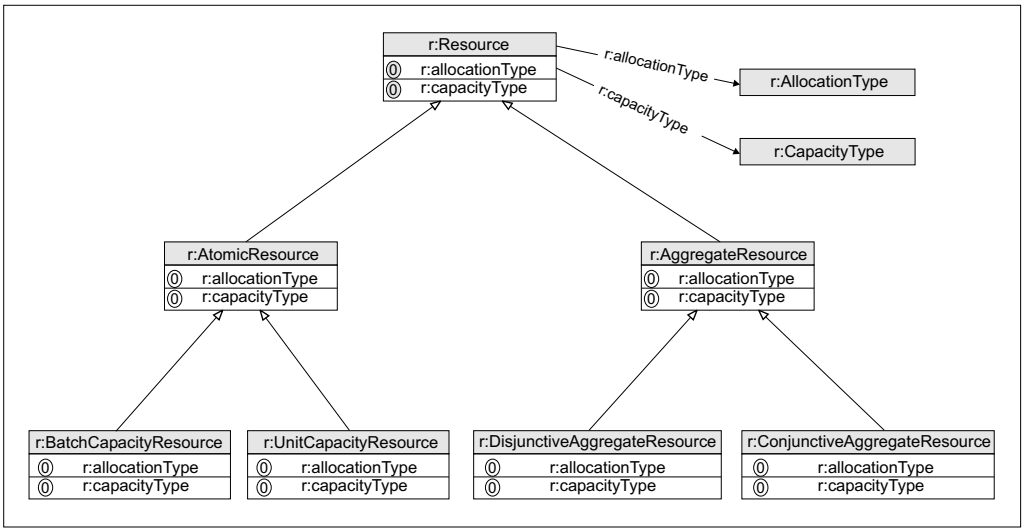

Figura 4.4: Ontologia de recursos de OWL-S, versão 1.0 [30], gerada pelo plugin ezOWL do Protégé 3.0.

4.2 Linguagem de Regras do Jena

Jena [62] é um framework Java desenvolvido pela HP (Hewlett-Packard) para a criação de aplicações para a Web semântica. Ele inclui, basicamente: (1) uma API para RDF; (2) uma API para OWL; (3) um

mecanismo de leitura e escrita de RDF em RDF/XML, N3 e N-Triples; (4) um mecanismo para armazenamento de dados em memória principal ou em memória secundária; (5) suporte à linguagem de consultas sobre dados RDF chamada RDQL; e (6) mecanismo de inferência. Atualmente, Jena está na versão 2.2.

A linguagem de regras do Jena foi escolhida para uso nesta tese, em lugar da linguagem SWRL associada à OWL-S, pois à época do início do trabalho não havia uma máquina de inferência para a linguagem SWRL. As regras escrita em Jena, por outro lado, podem ser facilmente executadas no próprio Jena, sobre uma base de dados em OWL, inclusive.

O mecanismo de inferência nativo do Jena [64] suporta inferência baseada em regras sobre grafos RDF (*Resource Description Framework*) e oferece três modelos distintos de execução: o modelo *forward chaining*, o *backward chaining* e o modelo híbrido. As configurações do mecanismo de inferência *GenericRuleReasoner* podem ser feitas através de seus parâmetros, dentre os quais o mais importante é o *PROPruleSet*, que determina o conjunto de regras que definem o seu comportamento.

Uma regra em Jena é definida pelo objeto Java *rule*, o qual contém uma lista de termos no corpo, que identificam as premissas, uma lista de termos na cabeça, que identificam as conclusões e um nome e uma direção opcionais. Cada termo, conhecido pelo objeto Java *ClauseEntry*, é o padrão de uma tripla, o padrão de uma tripla estendida ou uma chamada a um *builtin* primitivo. O Quadro 1 resume a sintaxe informal das regras Jena [64], onde as vírgulas são opcionais.

```
Rule      :=  bare-rule .
          or  [ bare-rule ]
          or  [ ruleName : bare-rule ]

bare-rule :=  term, ... term -> hterm, ... hterm // forward rule
          or  term, ... term <- term, ... term   // backward rule

hterm     :=  term
          or  [ bare-rule ]

term      :=  (node, node, node)      // triple pattern
          or  (node, node, functor)   // extended triple pattern
          or  builtin(node, ... node) // invoke procedural
                                      // primitive

functor   :=  functorName(node, ... node) // structured literal
```

```
node      :=  uri-ref             // e.g. http://foo.com/eg
          or  prefix:localname    // e.g. rdf:type
          or  ?varname            // variable
          or  'a literal'         // either a string or a number
          or  number              // e.g. 42 or 25.5
```

Quadro 1 - Sintaxe das regras em Jena.

Observe que Jena permite regras aninhadas. Além disso, novos *builtins* podem ser construídos e facilmente integrados ao *reasoner*, bem como novos *functors*.

Para permitir que as regras se tornem mais legíveis, Jena oferece suporte a URIs através de uma sintaxe simplificada. Para isso, basta que o usuário registre as URIs que necessita utilizando o objeto Java *PrintUtil*. Este objeto já conhece, por default, os *namespaces* de RDF, RDFS, OWL, DAML e XSD (*XML Schema*), devendo o usuário apenas registrar os *namespaces* adicionais necessários.

4.3 Resumo

OWL-S é definida como uma ontologia de serviços e uma ontologia de recursos. A ontologia de serviços trata composições de serviços como processos, por meio do *ServiceModel*.

OWL-S apresenta-se como uma boa alternativa para a representação de processos no âmbito dos sistemas de workflow que desejamos modelar. OWL-S possui as seguintes características: (1) inclui a definição de processos atômicos, compostos e simples (abstratos), pré-condições, e fluxo de dados; (2) separa claramente a lógica dos processos da tecnologia de implementação; (3) permite a expansão da ontologia no sentido da criação de novas classes e propriedades relativas a processos e recursos.

Nesta tese, estendemos OWL-S de forma a capturar características adicionais de processos e recursos, e dos relacionamentos entre eles, formando a base para definir o mecanismo de tratamento de exceção para a flexibilização, conforme discutido nos capítulos seguintes.